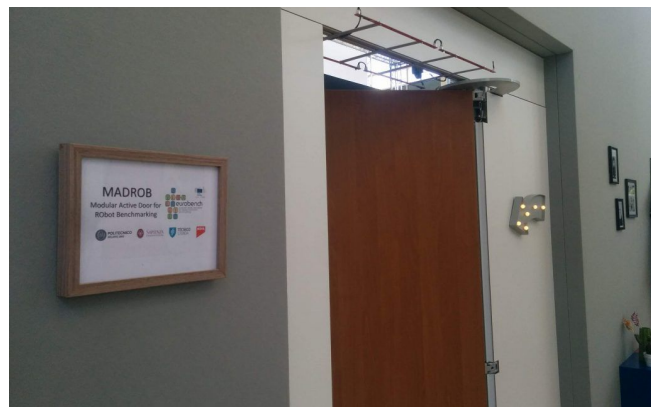
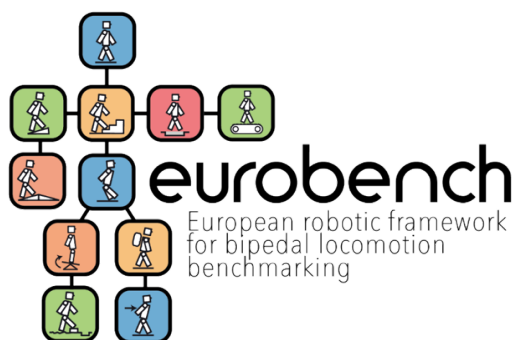
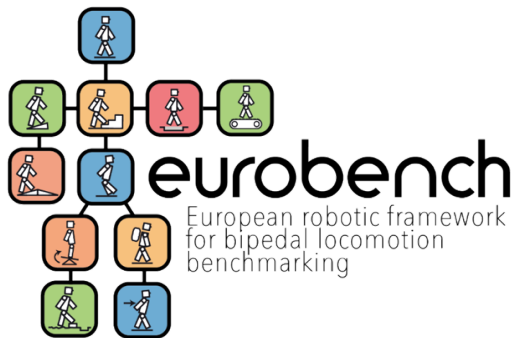


# MADROB Simulator Manual



Simulation environments are a fundamental tool to boost the research and the development of new methodologies and approaches within the robotics and AI community. In fact, having the possibility to gain access to a reliable simulation software that resembles a target task, and environment, allows researchers to explore a wide range of solutions and to remarkably reduce time and cost in the development process. The MADROB EuroBench project provides such a critical instrument. To this end, this manual provides a thorough overview of the EuroBench MADROB simulation suite: how to install, the overall architecture and how to run the simulation.

# Manual

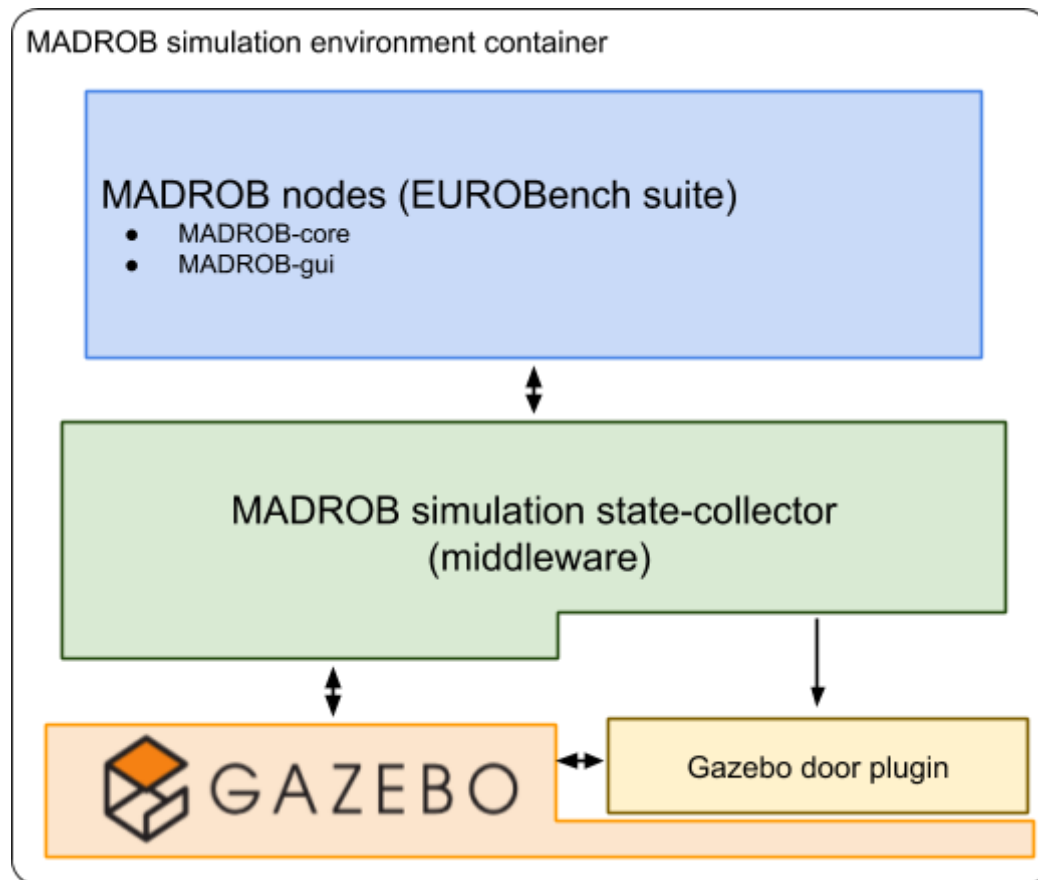


---

<b>Architecture</b>	<b>3</b>
Gazebo Door Plugin	3
MADROB Simulation State-collector Middleware	4
Communication with MADROB Nodes	7
<b>Installation</b>	<b>7</b>
Prerequisite	7
Instructions	7
Install Docker	7
Install nvidia-docker2	9
Clone and Build MADROB Simulator	9
Build ROS Packages	10
Installation Check	11
<b>How To Run</b>	<b>12</b>
How to Launch	12
How to change Simulation Configuration	13
Sensory Information Retrieval and Visualization	16
How to Replay Results	17

# Architecture

Before diving in the simulator features and installation guide, let us understand the overall architecture, its building blocks and data exchanged among different components. The figure below summarizes the architecture of the MADROB simulation environment.



## Gazebo Door Plugin

The gazebo door plugin contains the core computation routines to update the simulated door parameters and it exploits the ros-transport class to access ROS-nodes from an external source ([http://gazebosim.org/tutorials?tut=guided\\_i6](http://gazebosim.org/tutorials?tut=guided_i6)). Upon startup, the plugin receives all the initialization values needed to configure a particular run of the benchmark. The plugin is in charge of (1) assessing that all the ROS-nodes needed for the benchmark are operational; (2) setting the door angle limits (both door hinge and door handle); (3) set the opening side of the door; and (4) to load the door LookUpTable (LUT) containing the force torques to be applied to the door during the benchmark at run time.

Then, at each cycle of the simulation, it listens to the MADROB simulation middleware and waits for changes in the benchmark parameters such as the force torques LUT and the opening side of the door.

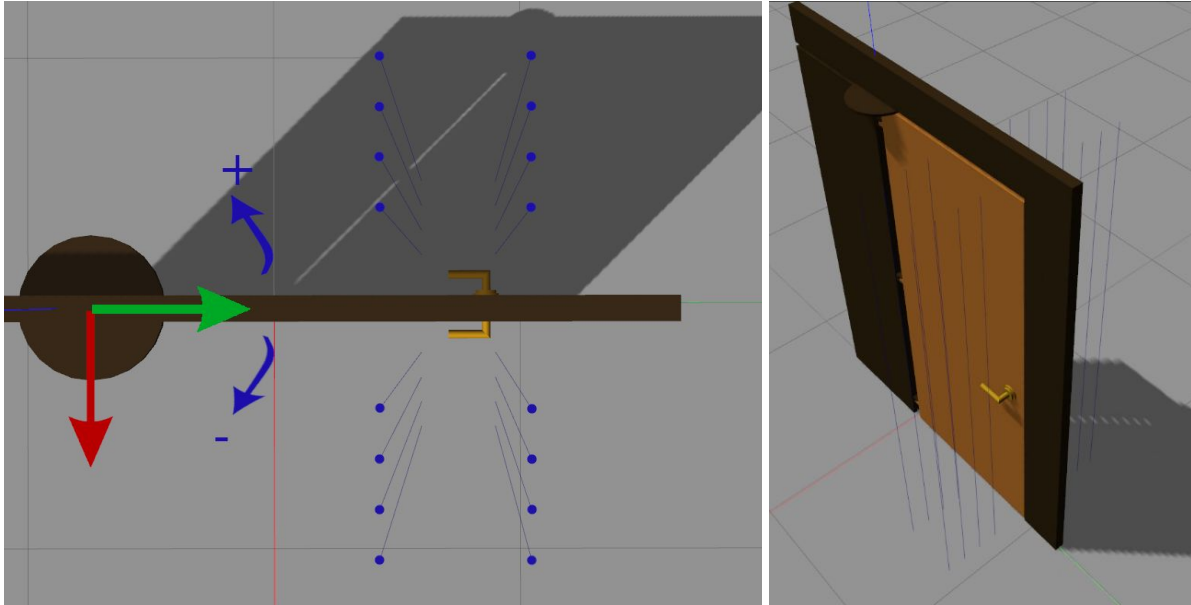
The door plugin communicates with both the Gazebo simulator and the MADROB simulation middleware. The former communication executes a constant and cyclic supervision of the door simulation. In particular, the simulator tells the plugin the current door opening angle, the plugin computes the force to be applied based on the provided LUT and returns the value to the simulator that sets the force to the door hinge. The latter communication is used instead to configure the benchmark and to wait for parameters changes. The plugin receives a message each time middleware sets a new torque LUT to be updated in a particular benchmark, and the opening side of the door and the robot approaching side. Both are required to set the direction to be applied to the force contained in the LUT.

## MADROB Simulation State-collector Middleware

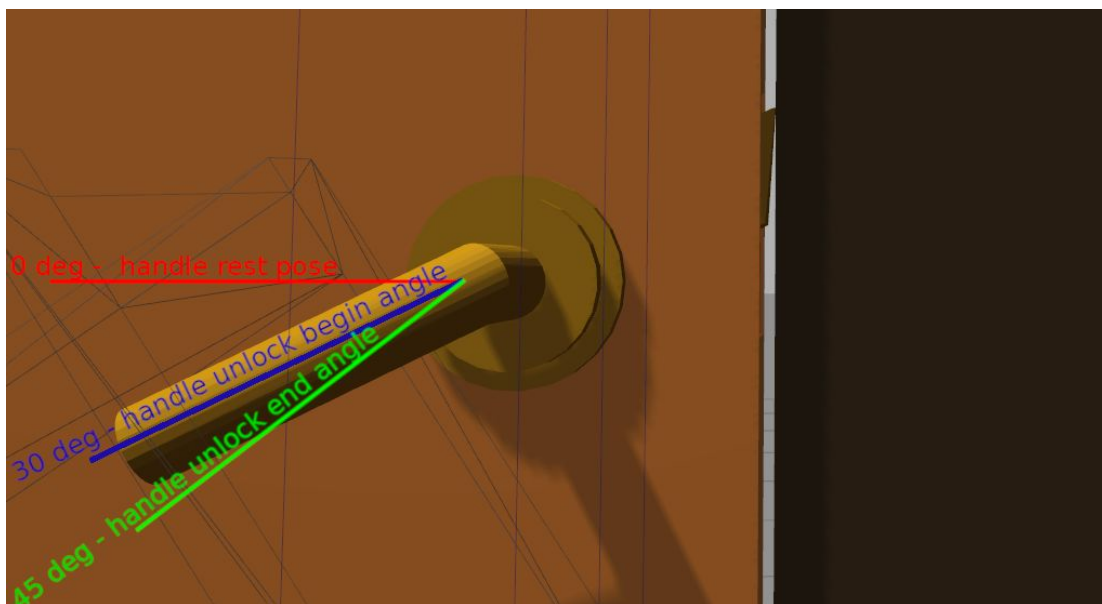
The middleware module is the core component of the simulation that makes the simulation environment properly configured to run Gazebo and makes all the information available to the MADROB evaluation suite in a compliant format. In fact, it is fundamental in order to achieve effective interchangeability between the real and the simulated robot. In fact, it provides all the information required by MADROB-core and MADROB-gui nodes ([https://github.com/madrob-beast/eurobench\\_benchmarking\\_software](https://github.com/madrob-beast/eurobench_benchmarking_software)) in the format imposed by the framework allowing the simulated benchmark to be controlled and evaluated via the official MADROB GUI.

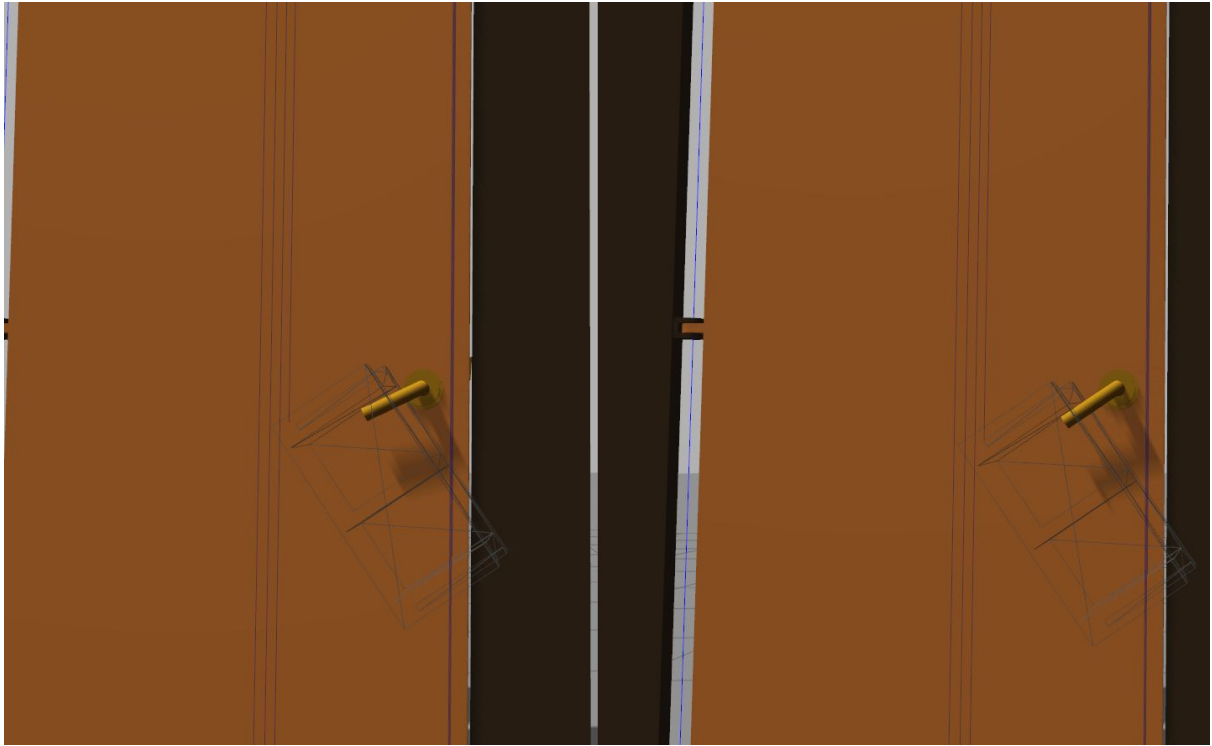
The middleware is in charge of configuring and starting the simulation components and receives from the simulator all the sensory information to be passed on to the evaluation suite. In particular, the environment simulates the 16 top-down position sensors placed above the door frame as in the figure below. The blue dots in the left image represent the origin of each sensor beam. It is worth mentioning that we built a custom sensors system of 16 infrared beams to emulate the real MADROB scenario. Each beam is configured as follows:

```
<update_rate>30</update_rate>
<ray>
  <scan>
    <horizontal>
      <samples>1</samples>
      <resolution>1</resolution>
      <min_angle>-0.001</min_angle>
      <max_angle>0.001</max_angle>
    </horizontal>
    <vertical>
      <samples>1</samples>
      <resolution>1</resolution>
      <min_angle>-0.001</min_angle>
      <max_angle>0.001</max_angle>
    </vertical>
  </scan>
  <range>
    <min>0.01</min>
    <max>2.2</max>
    <resolution>0.02</resolution>
  </range>
</ray>
```

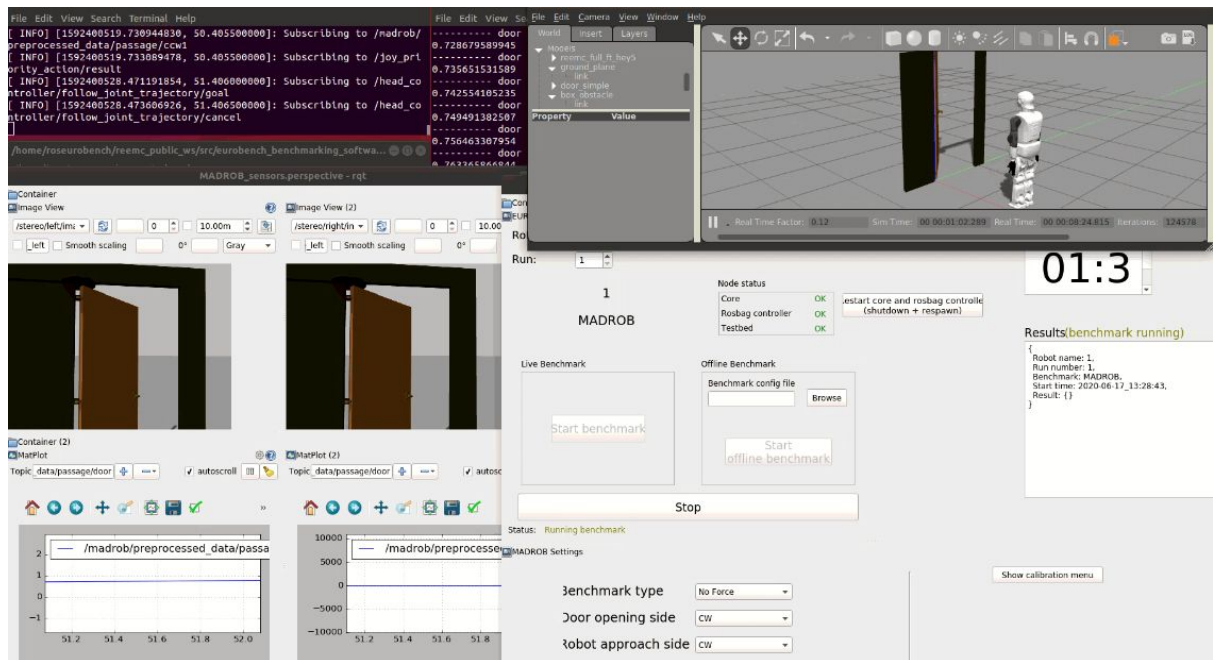


Moreover, it receives both the door hinge and handle angle. The figures below show the zero angle of both, on the left the door hinge when set to zero the door is closed and can reach -135 degrees or 135 degrees depending on the opening side (it is important to remember that the opening side changes according to the particular benchmark configuration). On the right the door handle with a zero rotation that can be operated by the robot and can be rotated up to until the door lock system is triggered (or activated). The handle is characterized by three reference angles: the handle rest pose (red in the figure below), the handle unlock begin angle at 30 degrees (blue) and the handle unlock end angle at 45 degrees (green) where the handle lock is completely open. This is important to emulate real handle lock systems that do not feature an instantaneous unlock of the door but in fact, the door plugin physically simulates a real door latch that needs to be operated.





The simulator also provides the robot camera images retrieved from the robot platform via the Reem-C gazebo plugin. It follows an example of images sampled from the robot camera stream while facing the MADROB challenge.



Finally, the middleware also communicates to the Gazebo simulator the starting position of the robot to be set that is relative to the door. The two positions that can be used are CW and CCW. The former represents a clockwise position of the robot with positive x-axis in the door reference frame, while the latter represents a counter-clockwise position of the robot with negative x-axis. Despite the side, both configurations place the robot one meter away

from the door. Furthermore, the middle communicates a restart interrupt to the simulator that allows for a quick reconfiguration of the active benchmark.

## Communication with MADROB Nodes

The last component in the simulation architecture is represented by the official MADROB framework. On this communication link there is a constant upstream flow of sensory information transferring all the data needed to perform the evaluation of the benchmark in the correct format required by the evaluation system. The communication link transfers the sensory readings in real time packaging on the proper messages, the 16 positional sensors, the door angle and the handle angle, and the robot camera stream.

At the same time it communicates downstream to the simulation environment events that require it to restart or to simply update the simulation parameters. As previously introduced, via the MADROB-GUI it is possible to configure the simulated benchmark by setting the force torques LUT, the door opening side and the robot approaching side. The figures below show the simulation benchmark initialized with the official MADROB GUI.

## Installation

### Prerequisite

The MADROB simulator runs in ubuntu and it is configured as a docker container. Such a setup makes the configuration and installation routine pretty straightforward, but on the other hand, introduces hard constraints on the hosting machine. In fact, the simulator can be installed, and it has been tested, on the following ubuntu versions:

- Ubuntu 16.04.7 LTS (Xenial Xerus)
- Ubuntu 18.04.5 LTS (Bionic Beaver)
- Ubuntu 20.04 LTS (Focal Fossa)

Moreover, the docker container extends the `osrf/ros:kinetic-desktop-full` docker (<https://github.com/craymichael/ros-kinetic-nvidia-docker>) that contains opengl, glvnd and cuda 8.0. This makes opengl work from any docker environment when used with `nvidia-docker2` (<https://github.com/NVIDIA/nvidia-docker>). Hence, the simulator can only run on a machine that features an NVIDIA graphic card that supports CUDA with the NVIDIA drivers installed.

## Instructions

### Install Docker

Docker can be installed by following the instructions in the official website (<https://docs.docker.com/engine/install/ubuntu/>). In particular, the simulation environment has been built with Docker version 19.03.13. In order to be completely self-contained, here we report the installation instructions that have been used to develop the simulator:



- Update the **apt** package index and install packages to allow **apt** to use a repository over HTTPS:

```
$ sudo apt-get update
$ sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent
software-properties-common
```

- Add Docker's official GPG key:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

- Verify that you now have the key with the fingerprint 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88, by searching for the last 8 characters of the fingerprint.

```
$ sudo apt-key fingerprint 0EBFCD88
pub rsa4096 2017-02-22 [SCEA]
    9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88
uid      [ unknown] Docker Release (CE deb) <docker@docker.com>
sub rsa4096 2017-02-22 [S]
```

- Use the following command to set up the **stable** repository

```
$ sudo add-apt-repository \
"deb [arch=amd64] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable"
```

- Update the **apt** package index, and install the *latest version* of Docker Engine and containerd, or go to the next step to install a specific version:

```
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

- Verify that Docker Engine is installed correctly by running the **hello-world** image

```
$ sudo docker run hello-world
```

This command downloads a test image and runs it in a container. When the container runs, it prints an informational message and exits.



## Install nvidia-docker2

The MADROB simulator builds upon the nvidia-docker2 that has to be installed (<https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html#docker>). Follow the installation instructions:

- Configure the stable repository and the GPG key:

```
$ distribution=$(. /etc/os-release;echo $ID$VERSION_ID) \  
&& curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo apt-key add - \  
&& curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-docker.list | \  
sudo tee /etc/apt/sources.list.d/nvidia-docker.list
```

- Update the apt package list and install the nvidia-docker2

```
$ sudo apt-get update  
$ sudo apt-get install -y nvidia-docker2
```

- Restart the Docker daemon to complete the installation after setting the default runtime:

```
$ sudo systemctl restart docker
```

- At this point, a working setup can be tested by running a base CUDA container

```
$ sudo docker run --rm --gpus all nvidia/cuda:11.0-base nvidia-smi
```

## Clone and Build MADROB Simulator

Now that the docker environment has been properly set up the simulator can be installed pretty easily. The MADROB simulation suite is self-contained in a docker that is also used as a workspace to gather all the MADROB nodes required to run the benchmark. All the setup and configuration is carried out in the Docker file in a way that once completed the docker container provides a link to a working simulation environment. By exploiting the docker paradigm all the step needed in order to install the simulator are to clone the repository and build the docker image with the provided script:

- Clone MADROB simulation repository

```
$ git clone  
https://github.com/madrob-beast/eurobench\_benchmarking\_simulation\_software.git
```

- Build the docker image

```
$ sudo ./build_the_container.sh
```

The container downloads and installs all the required components and it takes a few minutes to complete. Upon termination the script outputs the ID of the image (<ImageID>) that has to be used to generate the containers. For example, the figure below shows a successful build and the generated imageID is: 9de24e62610b.

```
---> 0b81e24000ea
Step 50/53 : RUN pip install PyYAML==5.1
---> Using cache
---> 0ea6c79c78a4
Step 51/53 : WORKDIR /home/roseurobench/reemc_public_ws/
---> Using cache
---> f5e27d50165d
Step 52/53 : RUN echo "\n\n\033[92m  DON'T FORGET TO RUN  catkin build -DCATKIN_ENABLE_TESTING=0 inside th
his procedure \n\n\033[0m "
---> Running in 8158f92aefeb

  DON'T FORGET TO RUN  catkin build -DCATKIN_ENABLE_TESTING=0 inside the docker
NB: the run script takes on input the id of this container tht you can find at end of this procedure

Removing intermediate container 8158f92aefeb
---> 7147d54ba051
Step 53/53 : SHELL ["/bin/bash", "-c", "source ../devel/setup.bash"]
---> Running in d33fc52b87ac
Removing intermediate container d33fc52b87ac
---> 9de24e62610b
Successfully built 9de24e62610b
Successfully tagged eurobench_nvidia_ros_reemc_madrob:latest
```

**NOTE.** Once the docker builds the image it is important to run the container and build the catkin workspace. The docker image also outputs the suggested command while building (green in the figure).

## Build ROS Packages

As anticipated, once the docker container is executed it is mandatory to run the `catkin build` command to properly configure the ros packages. In order to do so execute the container and run the catkin build command:

```
$ sudo ./run_the_container.sh <ImageID>
(docker-container)$ cd /home/roseurobench/reemc_public_ws/
(docker-container)$ source /opt/ros/kinetic/setup.bash
(docker-container)$ catkin build -DCATKIN_ENABLE_TESTING=0
```

**NOTE.** The run outputs the container ID <ContainerID> to be used in order to execute the newly built image.

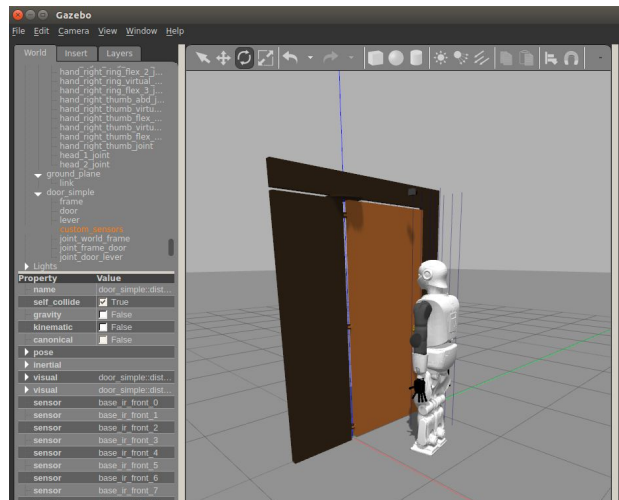
## Installation Check

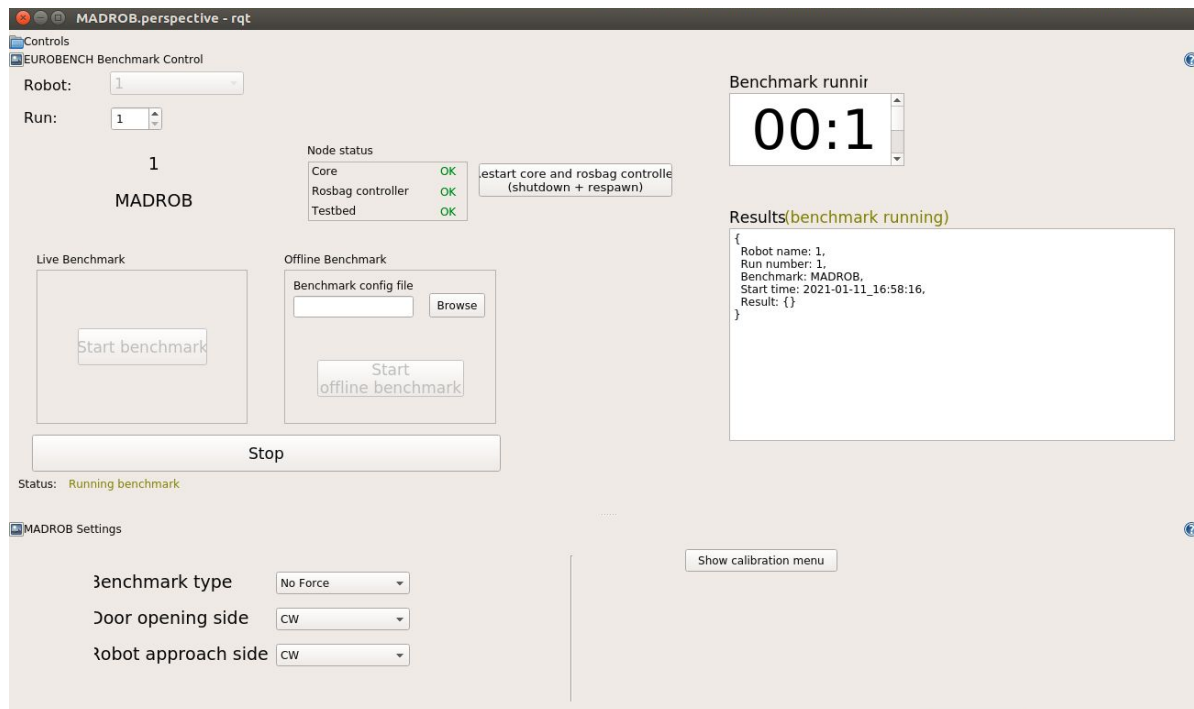
Check that everything is installed correctly by launching the simulator. To this end run the script that you find in the root of the repository by inserting the image ID printed out at the build phase. Once the ROS packages have been compiled you can open a different terminal and attach from it by using the `<ContainerID>`.

```
$ sudo docker exec -it <ContainerID> /bin/bash
(docker-container)$ source /home/roseurobench/reemc_public_ws/devel/setup.bash
(docker-container)$ QT_X11_NO_MITSHM=1 roslaunch eurobench_reemc_door
reemc_door.launch
```

The roslaunch command executes the simulation environment and shows the Reem-c robot facing a door. If everything has been installed correctly this is what should be visualized.

**NOTE.** `QT_X11_NO_MITSHM=1` is mandatory and tells QT to send the screen of the simulation, throughout the proper channels, outside the container.





## How To Run

### How to Launch

The use of docker containers make the use of the simulator safe and intuitiv. In fact, once it has been properly installed, the simulator makes its execution run seemingless on any machine. To properly launch a simulation of the benchmark we need to go through three steps. First, we need to launch the MADROB core nodes. Second, we need to launch the simulator node that takes care of setting up both the MADROB simulation middleware and the gazebo simulator with MADROB door-plugin. Lastly, we need to launch a monitoring node that visualizes the sensory information coming from the simulator.

Hence, to launch the MADROB core nodes open a terminal and gain access to the docker container previously installed.

If it is the first time and you need to create the container open a terminal and run

```
$ sudo ./run_the_container.sh <ImageID>
(docker-container)$ cd /home/roseurobench/reemc_public_ws/
(docker-container)$ source /opt/ros/kinetic/setup.bash
(docker-container)$ catkin build -DCATKIN_ENABLE_TESTING=0
(docker-container)$ roslaunch eurobench_benchmark_core madrob_all.launch
```

Else, if a container of the MADROB simulator has already been created you can attach to the container from a different terminal and launch the MADROB core by using the following instructions

```
$ sudo docker exec -it <ContainerID> /bin/bash
(docker-container)$ roslaunch eurobench_benchmark_core madrob_all.launch
```

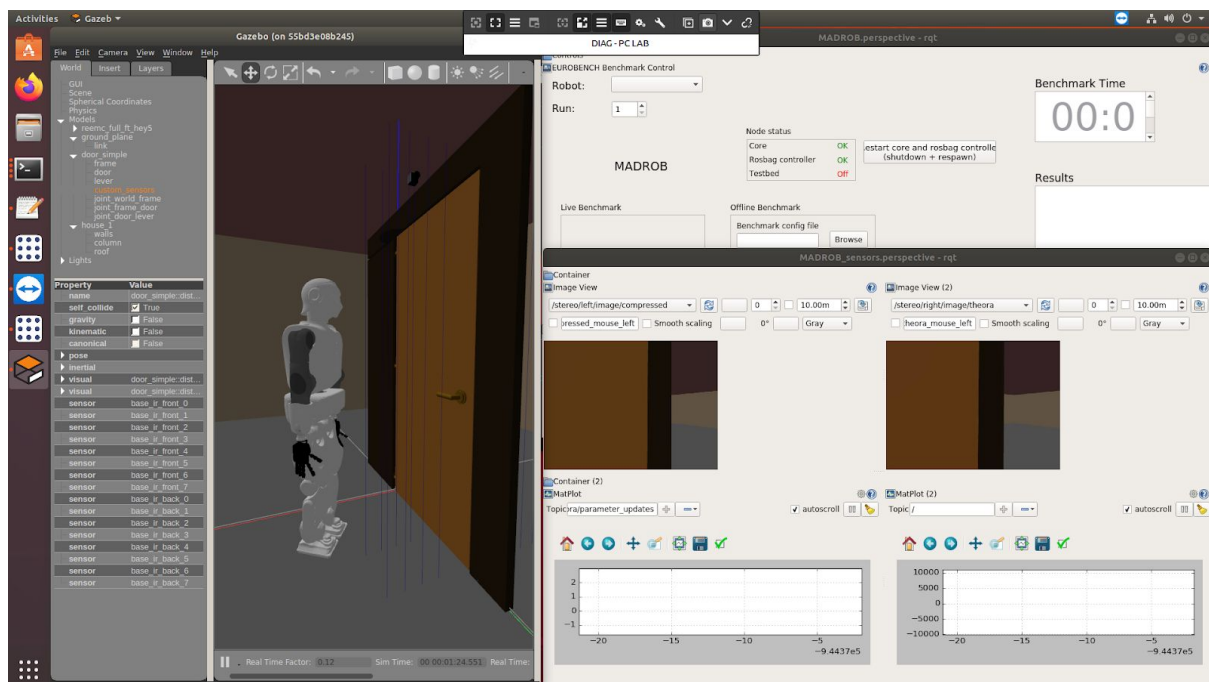
The second step launched the actual simulation. To this end, open a new terminal and attach to the same container. Then launch the simulation

```
$ sudo docker exec -it <ContainerID> /bin/bash
(docker-container)$ QT_X11_NO_MITSHM=1 rosrn madrob_simulation_state_collector
madrob_simulation_state_collector_node.py
```

Finally, to run the sensor GUI interface to monitor the robot camera stream and visualize the benchmark sensors, open a third terminal and run

```
$ sudo docker exec -it <ContainerID> /bin/bash
(docker-container)$ roslaunch benchmark_gui madrob_sensors_gui.launch
```

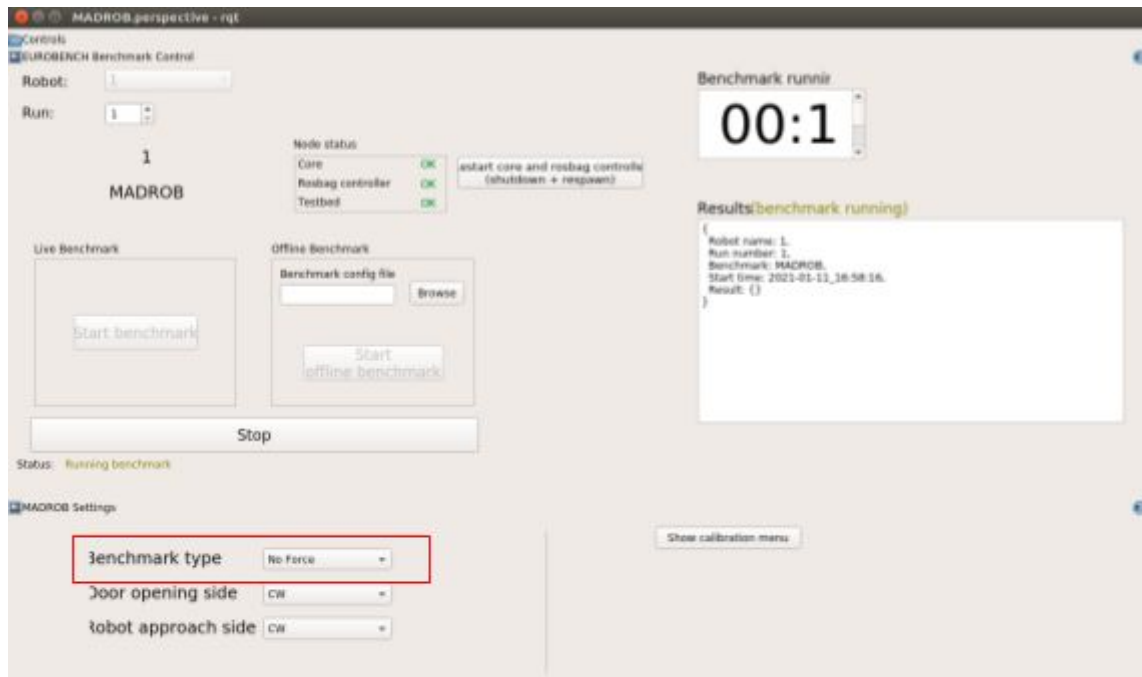
Once the procedure is complete both the MADROB official GUI and the Gazebo simulated Reem-C robot appear to screen and are ready to use. The figure below shows the simulation benchmark in idle



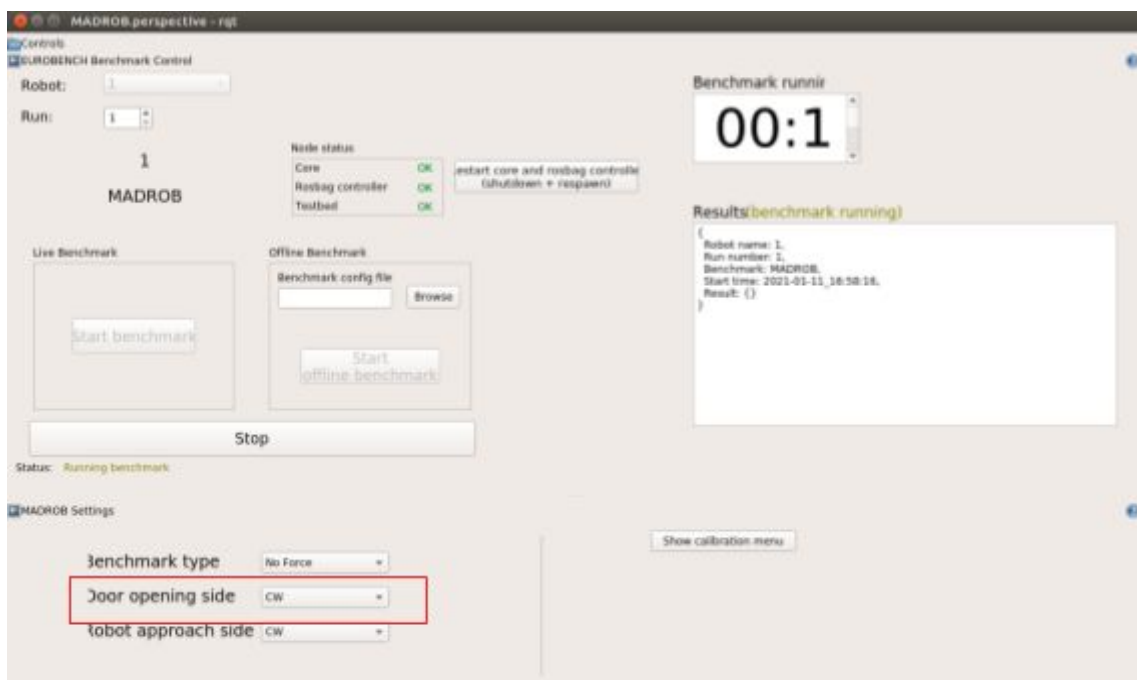
## How to change Simulation Configuration

As for the real-robot benchmarks the simulations configuration can be changed through the MADROB GUI. Suggestively, it is possible to reconfigure: (1) the door controller and update the LUT of the force torques profiles; (2) the door opening side (CW and CCW); and (3) the

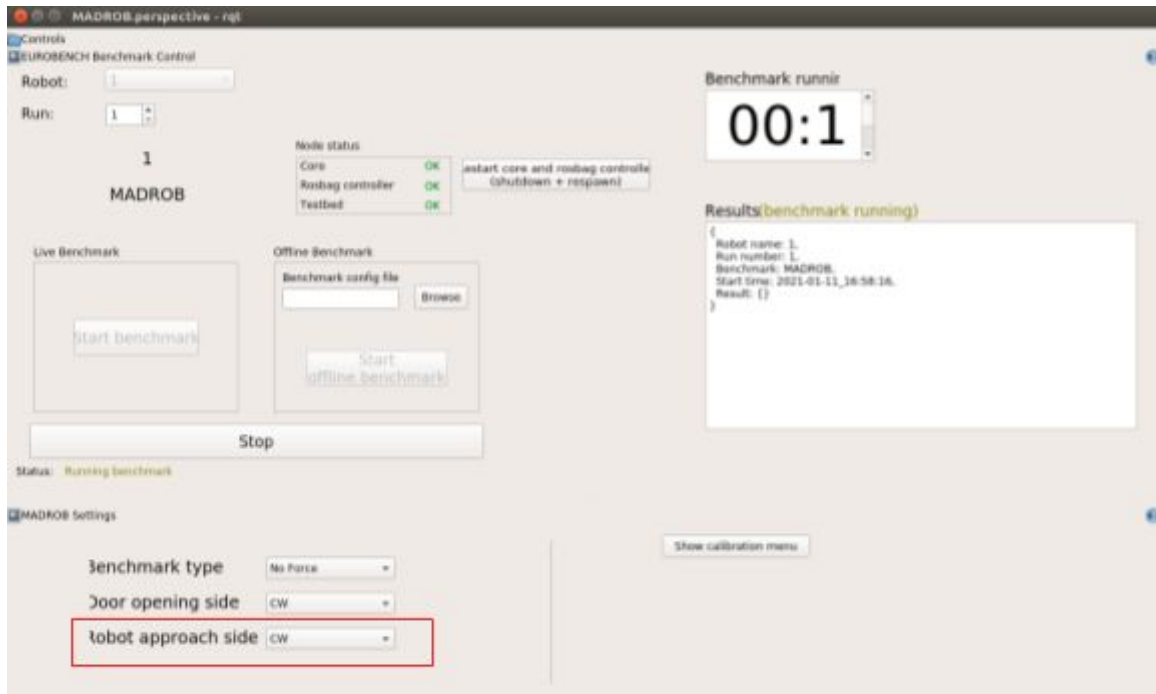
robot approaching side (CW and CCW). In the next three figures we can respectively see highlighted in red who to change the aforementioned parameters.



Benchmark type



Door opening side



Robot approaching side

In the door benchmark configuration bracket (first figure) we can select from a list of available force profiles, namely:

- No Force
- Constant Force
- Sudden Ramp
- Wind Ramp
- Sudden Force

It is important to note that this action does not stop nor restart the simulation and updates the force profiles online.

Note: In order to edit (or add) new force configurations, it is needed to edit and recompile the gazebo plugin *simple\_door\_config*. In particular, the force profile lut values are located at “eurobech/reemc\_door/plugins/LUT.h” in

[https://github.com/madrob-beast/eurobench\\_reemc\\_door](https://github.com/madrob-beast/eurobench_reemc_door). LUTs are stored as vectors and are specified for each side of the door.

The second figure shows how to change the door opening side. There are two possible choices that, however, require the simulation to restart in order to not invalidate the simulation results. Hence, once this door opening side is changed, the gazebo simulation stops and then restarts.

Note: In order to modify the maximum door opening angle, one has to edit and recompile the gazebo plugin “*simple\_door\_config.cc*” in

[https://github.com/madrob-beast/eurobench\\_reemc\\_door](https://github.com/madrob-beast/eurobench_reemc_door). In particular, the door hinge angles are set in the “onLoad” Gazebo callback.

The last controllable parameter is the robot approaching side (see third figure). In both cases, CW and CCW, the robot position is reset to be one meter far from the door frame. As



before, if this parameter is changed, the simulation stops and a new scene is launched afterwards.

Finally, it is important to understand that the simulation can begin only after the robot number is selected (in the upper left part of the GUI). This will be stored in the logging files. The simulation can always be controlled using the Start/Stop button and once the benchmark is concluded results are stored in the docker container at `~/eurobench_output/` folder (or equivalently `/root/eurobench_output/`).

Note: the docker container has the `/root/` in the same folder of the `/home/` directory. Hence, even if the MADROB-core saves results in the home directory, the container saves them in the root due to the docker container file system.

## Sensory Information Retrieval and Visualization

The simulation comes with a visualization node that allows for a visualization of the sensor observations, namely (1) the 16 raw range distance sensors placed on the door frame; (2) the interpreted distance data compliant with the MADROB message topics guideline template; (3) the door hinge and handle opening angle; and (4) the robot camera images. For each of these sensor streams coming from the simulator node, the data flows on standard ROS topics. Topic names are listed below:

- Raw measures of the 16 distance sensors placed above the door frame:

Data type: `sensor_msgs/Range`

```
/sensor/base_ir_front_0  
/sensor/base_ir_front_1  
/sensor/base_ir_front_2  
/sensor/base_ir_front_3  
/sensor/base_ir_front_4  
/sensor/base_ir_front_5  
/sensor/base_ir_front_6  
/sensor/base_ir_front_7  
/sensor/base_ir_back_1  
/sensor/base_ir_back_2  
/sensor/base_ir_back_3  
/sensor/base_ir_back_4  
/sensor/base_ir_back_5  
/sensor/base_ir_back_6  
/sensor/base_ir_back_7
```

- The distance sensor message is provided by the `madrob_msgs` package within the MADROB core suite. The aforementioned simulated range sensors are thus processed to generate ROS topics that are compliant with the `madrob_msgs/Passage` message and the MADROB guidelines. Post-processed information, as for the real-robot benchmark, are streamed on the following topics:

Data type: `madrob_msgs/Passage`

```

/madrob/passage/ccw_left
/madrob/passage/cw_left
/madrob/passage/ccw_right
/madrob/passage/cw_right

```

- Door and handle opening angles:

```

Data type: std_msgs/Float64
/madrob/preprocessed_data/passage/door
/madrob/preprocessed_data/passage/handle

```

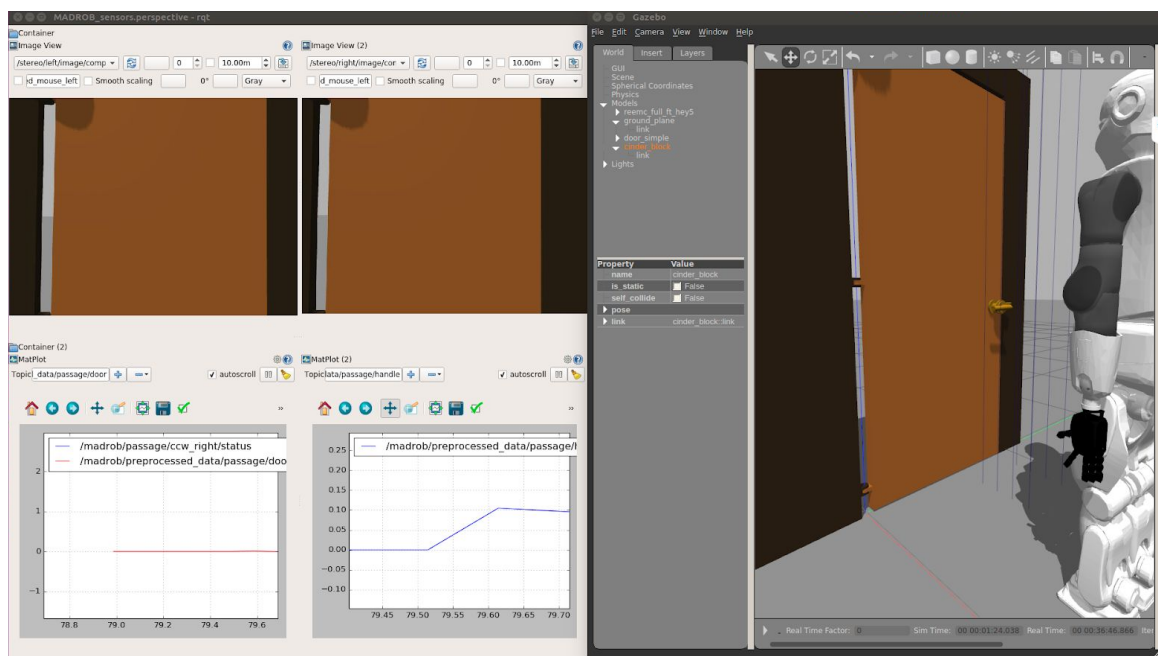
- Image streaming from left and right cameras:

```

Data type: sensor_msgs/CompressedImage
/stereo/left/image/compressed
/stereo/right/image/compressed

```

The sensory data can be visualized with the eurobench MADROB\_sensor\_perspective. By using this tool it is possible to have realtime camera streams and sensor data plots. The two camera plots can be chosen in the upper drop down menus of the GUI. All the remaining sensory data flows can be added to the graphical interface by choosing the topic in one of the two selector and by pressing the plus sign. The figure below shows an example.



## How to Replay Results

In order to replay a particular run of the benchmark, locate the desired output in `~/eurobench_output/`. The figure below shows an example of the content in the output

directory. Here are listed three different runs of the benchmark. The single digit folder `<ROBOT-NUM>` contains a .txt file that stores information about the robot used for the experiment and the benchmark configurations parameters and statistics. The GUI allows the selection of two types of robots, hence there can be a maximum of two folders labeled with a digit number. Conversely, the folder named `MADROB_<ROBOT-NUM>_<EXP-ID>` contains information related to each individual run of the benchmark. In each folder there is a .yaml needed to index and replay the benchmark, a ROS bags file with all the topic sensory data, and two .csv files storing the events occurred during the simulation, and the record of the applied forces.

```
root@2328ed982cdc:~/eurobench_output# ls -lla
total 32
drwxr-xr-x 7 root root 4096 Jan 15 15:42 .
drwx----- 1 root root 4096 Jan 15 15:40 ..
drwxr-xr-x 2 root root 4096 Jan 15 15:37 1
drwxr-xr-x 2 root root 4096 Jan 15 15:42 2
drwxr-xr-x 2 root root 4096 Jan 15 15:24 MADROB_1_001_20210115_152455
drwxr-xr-x 2 root root 4096 Jan 15 15:41 MADROB_2_001_20210115_154057
drwxr-xr-x 2 root root 4096 Jan 15 15:42 MADROB_2_001_20210115_154221
root@2328ed982cdc:~/eurobench_output#
```

The .yaml file inside the folder `MADROB_<ROBOT-NUM>_<EXP-ID>` can be used to replay offline simulation experiments, as in the case of real-robot benchmark execution. To this end, it is possible to input the full path to the .yaml file of a particular experiment in the *Offline benchmark* section of the madrob.perspective.GUI and then press the *Start offline benchmark* button. This is highlighted in the image below.

